

## 图文 049、第7周答疑：本周问题答疑汇总

700 人次阅读 2019-08-18 09:02:02

[详情](#) [评论](#)

### 第7周学员答疑汇总

#### 学员总结：

就着这篇作业，总结一下。我理解的优化思路：项目上线初期：

1、上线前，根据预期的并发量、平均每个任务的内存需求大小等，然后评估需要使用几台机器来承载，每台机器需要什么样的配置。

2、根据系统的任务处理速度，评估内存使用情况，然后合理分配Eden、Survivor，老年代的内存大小。

总体原则就是，让短命对象在YoungGC就被回收，不要进入老年代，长期存活的对象，尽早进入老年代，不要在新生代复制来复制去。对系统响应时间敏感且内存需求大的，建议采用G1回收器

如何合理分配各个区域:

根据内存增速来评估多久进行Young GC

根据每次Young GC的存活，评估一下Survivor区的大小设置是否合理

评估多久进行一次FullGC，产生的STW，是否可以接受？

公司的运营很牛，过了一段时间，系统负载增加了10倍，100倍：

方案1：增加服务器数量 根据系统负载的增比，同比增加机器数量，机器配置，和jvm的配置可以保持不变。

方案2：使用更高配置的机器 更高的配置，意味着更快速的处理速度和更大的内存。响应时间敏感且内存需求大的使用G1回收器 这时候需要和‘项目上线初期’一样，合理的使用配置和分配内存

**回答：**理解完全正确

#### 问题：

老师，如果在做定时任务数据处理的时候，在一个方法里面通过多线程的方式来处理数据库中的数据，在其他线程没有结束之前，其中一个线程提前处理完了，那该线程所new的对象会变成垃圾对象吗？还是说也要等到其他线程都处

理完了才会变成垃圾对象？

**回答：**他自己引用的对象，他自己线程销毁了，自然那些对象都是垃圾了，如果触发gc就会被回收

**问题：**

在main方法中周期性执行loadReplicasFromDisk方法，文中说，loadReplicasFromDisk方法执行完毕之后，一旦方法结束，loadReplicasFromDisk方法的栈帧就会出栈，就没人引用ReplicaManager对象了。

我的疑问是为什么loadReplicasFromDisk执行完毕就会出栈呢，不应该是一直调用这个loadReplicasFromDisk方法，对应的一直有新的loadReplicasFromDisk方法的栈帧入栈

除非while true结束了，也就是main方法结束了，各个栈帧才会按照后进先出的顺序陆续出栈吗？栈帧A出栈后，才会回收栈帧A的局部变量表，这个时候才可以说没人引用栈帧A创建的那个ReplicaManager对象。

**回答：**一次方法调用就得入栈一次，一次方法调用结束了就会出栈，不是说一个方法调用多次，他就一直在栈里的

**学员总结：**

我们线上的系统 young gc 和 full gc 都比较频繁，但是业务层面并不要求实时性很大

学完一系列文章以后 偷偷的在服务器上加上了 parnew cms收集器 然后差不多7分钟一次young gc 然后几个小时一次full gc 但是贷款业务真的对延时不在乎

**回答：**非常好，无论在乎不在乎，你自己优化一下，以后面试和工作都需要用到这个jvm优化技能

**问题：**

老师，我想问一下，当一次MinorGC时 发现应该进行 FullGC，此时就进行FullGC。今天的我课中说到，在并发清理的过程中可能会有Minor 送来的新对象在老年代放不下，就会出发 Concurrent Mode Failure 错误。

我想问的是： 在第一次的Minor 触发FullGC时， Minor 送来的那批对象是怎么处理的呢？

是先卡一会儿，等 FullGC的并发清理阶段 有了那么大的空间后，这次的Minor对象就放下去，然后这次MinorGC结束。此时 FullGC依旧在处理并发清理阶段，这个时候若老年代进来的对象大于可用的连续内存，所以就换成 Serial Old 去 STW 单线程的 GC 全部的老年代，然后在把这批对象放进去是吗？

**回答：**是的，你理解没错

**学员总结：**

文章写的太好了，刚买几天，太精彩了，这个周末忍不住一口气把42章全看完了，基本上都理解了，就是东西太多有点记不住，还得从新慢慢看一下总结一波才行，总算是理解jvm这东西了

之前看书看的晕头晕脑得，老师讲的非常通俗易懂，再加上这么案例的讲解，以前一直认为jvm优化有啥用呢，工作中根本用不到，看完案例才知道这个东西的重要性，感谢老师！

**回答：**多谢支持，继续加油

**问题：**

有个疑问：每次1w条数据需要处理10秒，那么每分钟也就处理6次数据，那么新生代就会积压94次数据，即积压940M（94\*10M）数据未处理。也就是说1分钟把新生代填满，940M存活。而不是文中说的200M存活对象啊。

如果老师指的10秒的处理时间是单核处理速度，那每分钟4核也就处理240M还剩760M存活呢？

**回答：**不是的，1分钟来计算，填满之后，会垃圾回收，大部分对象会被回收掉，然后遗留200MB左右的数据是存活的

**问题：**

老师，在你的例子中，把s区扩大到200M，刚好占满s区，然后就只有很少的对象进入老年代。

我有个疑问，都占到s区的100%了，难道不会触发年龄担保机制么？

**回答：**对的，这里我只是做一个示例，其实最好让S增加到400左右，避免触发动态年龄判定机制

**问题：**

我有两个问题:

1.基于动态年龄来判断的时候，如果年龄1-5的对象占有s区的50%，那么是 > = 5还是 > 5的对象会进入老年代？这里是否会有 = 号？

2.按照动态年龄判断的规则，是不是每次minor gc后，都保证s区存活的对象  $\leq$  s区大小的50%?(因为根据年龄判断，50%以外的那部分直接进入老年代了，所以每次minor gc后s区最多剩下50%)

**回答：**

1、 $\geq 5$

2、对的，你最好是让存活对象小于S的50%

**问题：**

这里我有个误区，那就是动态年龄的判断并不伴随minor gc，他应该拥有属于自己的触发时机(何时会触发?)

那我是不是可以这么理解：基于动态年龄的机制是随时有可能会触发的(那么触发的时机是什么?)

这时候，s区剩余空间一定是  $\leq 50\%$ 。但是minor gc之后，s区的存活对象是有可能  $> 50\%$ 的，只有等到下一次动态年龄判断触发时，又会下降到50%及以下。

**回答：** 每次Minor GC过后就会触发动态年龄判定机制

**学员总结：**

参数设置： 参数设置了堆大小为10M，指定了新生代5M，那么老年代也是5M，因为SurvivorRatio=8，所以Eden区4M，S1、S2分别是0.5M。

GC前的空间分配及判断： 先分配了3次1M对象进入Eden区，再次分配2M时，Eden区放不下了，准备进行Young GC，此时需要判断是否需要空间担保，5M的老年代空余 $> 3$ M的新生代已用空间，不需要空间担保。直接进行Young GC。

执行Young GC： 遍历所有线程栈中的和静态的引用变量，进行GCRoots追踪，发现新生代的Eden和S1区没有对象存活，不需要进行对象的拷贝，直接清空Eden区和S1区。此时YoungGC执行完成。

**回答：** 对的

**问题：**

问题1：案例中的s区大小为200m，刚好容得下每次minor gc后存活对象的大小，但是根据动态年龄判断的规则

因此这里的s区大小是否应该设置为400m，才能更大程度的避免对象在一次minor gc后进入老年代？（也就是调整eden:s:s=6:2:2）但是这样好像又导致了更加频繁的minor gc

问题2：文章中在提及动态年龄判断规则的时候，说的是同龄对象占有s区大小> 50%，但是其实应该是年龄1-N的对象占有s区大小>50%吧，然后年龄>=N的对象会直接进入老年代

**回答：**

1、那个是示例，其实S可以搞的再大点，让存活对象小于S区的50%

2、对的，后面有多处说明，就是你说的这个意思

**学员总结：**

新生代初始5M，最大5M，堆初始10M，总大小10M。Eden和Survivor区比例 8:1:1，即 4M:0.5M:0.5M，对象超过10M直接在老年代。

垃圾回收算法: 新生代 ParNew + 老年代 CMS

第一次分配 1M 第二次第三次第四次, 失去指向, 存在3M垃圾, Eden还剩 1M可用空间

分配 array2 为 2M Eden区空间不够

需要对新生代进行youngGC

回收垃圾大小为3M 和老年代剩余连续空间大小作比较 3M < 5 M, 不需要空间担保和Full GC

新生代进行垃圾回收 STW

将array2 放到 Eden区

**回答：**总结的很好

**学员评价：**

读完这节，以及本周的安排 我意识到老师的课程设计是我见过的最佳之一。 为什么这么说呢？

因为课程不仅由浅入深 一步一图，更是考虑到了 学而时习之 周期性回顾的重要性，节奏比速度更重要。

可能有些老师会忽略一件事：就是以为学生的精力都放在他这一门课上，其实不是的

以我为例，我同时在学jvm、算法、石杉的架构(此处硬广)，精力那是要平衡分配的，如果某一门课一味求快 求多，可能最终是学完就忘，欲速则不达。

**回答：**是的，其实专栏类的东西，最好的就是在全流程里贯穿核心的知识体系，反复强化，不停的强化，适当的复习和总结，最终一个长周期后，彻底掌握这个技术的实战

## 学员总结：

整个堆的情况 Eden : 4m Survivor : 0.5m old : 5M

正常来说的过程应该是这样的：开始分配3m的对象，有2m是垃圾对象，此时分配2m的对象时发现eden空间不够，根据空间担保规则，老年代5m大于新生代已占用内存3m，所以放心MinorGC，GC完了之后，新生代还有1M对象存活(array1引用)

此时发现survivor的一个区域0.5m不足以放下存活的对象会将1m对象转移到老年代，接着新生代开始分配2m的对象

对象分配完毕之后整个堆的情况：新生代2m，老年代1m。事实上，经过我多次测试，运行这样的代码会产生2次MinorGC和一次Full GC。

这个结果可能根据每个人的电脑不一样也会产生误差(也有可能是在IDEA运行的原因)。我的JDK版本为1.8.0.201

另外根据GC日志分析，其实他计算的不单单是我们程序使用的内存，可能jvm本身使用的内存也计算上了。

虽然我们heap大小设置为10m,但是GC日志中只有9728k，并不是10m，所以我认为在分析GC的时候，还是无法精确分析到每一个细节的，我们只能大致上来猜测GC的行为。

## 学员思考题回答：

深夜打卡。

1、老年代5m空间大于3m，可以直接YoungGC，3M垃圾直接回收，Eden区清空，S1，S2也清空。但实际情况，根据日志，from space是占满了。

2、根据日志，发现确实进行了一次Young GC。具体日志：0.116: [GC (Allocation Failure) 0.116: [ParNew: 3964K->512K(4608K), 0.0009533 secs] 3964K->549K(9728K), 0.0010680 secs] [Times: user=0.05 sys=0.00, real=0.01 secs]

其中Allocation Failure应该表明了GC的原因（连续内存空间不足，内存空间分配失败），和具体的回收情况，可以看到内存空间被释放出来了，但却不是全部释放（可以看到原本的空间是3964K，而不是就三个数组的3072K）

**回答：**非常好

**问题：**

老师，请教一个问题，就是survivor区使用达到100%，按之前的规则一批对象占用survivor区超过50%，年龄大于等于这批对象的就进入老年代，好像这次gc没有触发，我要怎样才能模拟出这个规则的触发？

**回答：**他这一次达到Survivor 100%不会立马触发动态年龄判定机制，需要下一次GC的时候看你还是超过Survivor 50%，才会进行动态年龄判定，往后看，会有演示的

**问题：**

Java8 取消了PermGen。取而代之的是MetaSpace，方法区在java8以后移至MetaSpace。Jdk8开始把类的元数据放到本地内存（native heap），称之为MetaSpace

理论上本地内存剩余多少，MetaSpace就有多大，当然我们也不可能无限制的增大MetaSpace，需要用-XX:MaxMetaSpaceSize来指定MetaSpace区域大小。

关于used capacity committed 和reserved，在stackoverflow找到个比较靠谱的答案，我尝试翻译一下：  
MetaSpace由一个或多个Virtual Space（虚拟空间）组成。虚拟空间是操作系统的连续存储空间，虚拟空间是按需分配的。当被分配时，虚拟空间会向操作系统预留（reserve）空间，但还没有被提交（committed）。

MetaSpace的预留空间（reserved）是全部虚拟空间的大小。虚拟空间的最小分配单元是MetaChunk（也可以说是Chunk）。

当新的Chunk被分配至虚拟空间时，与Chunk相关的内存空间被提交了（committed）。MetaSpace的committed指的是所有Chunk占有的空间。

每个Chunk占据空间不同，当一个类加载器（Class Loader）被gc时，所有与之关联的Chunk被释放（freed）。这些被释放的Chunk被维护在一个全局的释放数组里。

MetaSpace的capacity指的是所有未被释放的Chunk占据的空间。这么看gc日志发现自己committed是4864K，capacity4486K。有一部分的Chunk已经被释放了，代表有类加载器被回收了

可以这么理解吗老师？

附上原文链接：

<https://stackoverflow.com/questions/40891433/understanding-metaspace-line-in-jvm-heap-printout>

有一个示意图 有助于理解

**回答：**非常好，标准答案，优秀

**学员总结：**

这周实际操作的课程，只有周末才有时间试试。懂了原理，会看 GC 日志，感觉这个垃圾回收的原理，基本没问题了。

**回答：** 对的，吃透这些，jvm运行原理基本没问题了

**问题：**

可是minor gc不是会产生stw吗？ 那么stw是允许仅所有的垃圾回收线程运行？

即使是两个不同的正在运行的垃圾收集器，他们也是可以同时运行的，只不过工作线程无法一起运行罢了。请问是这样吗？

**回答：** stw，是说停止所有工作线程，不是垃圾回收线程

**问题：**

文中第二次发生gc的时候，清除了eden区域的对象，并判断from survivor的一岁的对象大于50%。然后准备将survivor所有对象移动至老年代。然后第三次gc的时候，将survivor全部对象移动至老年代

但是为什么此时GC日志写的是（Allocation Failure）分配失败呢？

并且新生代那7017K的对象是哪里来的呢？不是前面一次gc后只剩下713K吗？期间就算有新对象生成也应该只有array4的2M对象？

求老师解惑

**回答：**

1、Allocation Failure，意思就是分配对象的时候内存不够触发了gc

2、7000kb，不是文章里解释了吗，有一些是对象头，还有一些是未知对象

**问题：**



关于动态年龄判断的疑问：ParNew: 7017K->0K，这次Young GC 没有Eden区的对象存活，此时Survivor区的存活对象年龄+1 变为2岁，按照文中的动态年龄判断逻辑不是应该“大于2岁”的对象进入老年代么？

如果按照专栏25讲描述的动态年龄判断逻辑倒是可以解释的通（一批同龄对象，直接超过了Survivor区空间的50%，此时也可能会导致对象进入老年代）

**回答：**对的，他就是同龄对象超过了Survivor的50%，所以进入老年代了

**问题：**

请问老师后续规划的 mysql 实战专栏，会包括分库分表部分吗？

**回答：**那是肯定的

**问题：**

老师，您好，请问，是不是不仅仅新生代eden区满了，会回收新生代，有时候，如果G1觉得region的个数可以满足200ms的时候，也会回收？那回收的是新生代吗？还是mixgc

**回答：**是的，g1感觉可以回收了也会回收，通常是新生代

**问题：**

老师，我接着上一条评论发文。可能我说的不够清晰

我想问的是 对象移动至老年代的时候发生了两次GC。第一次GC是7260k--713k。然后紧接着又发生了一次GC是7017k--0k。

第一次GC是因为为array4分配内存的时候分配失败发生了一次gc。但是为什么紧接着又发生了一次GC？并且原因也是分配失败，明明堆内存此时只占用713K，是分配未知对象的时候分配失败了吗？而且这将近7M的对象都是未知对象吗？就算此时array4分配内存也才占用2M而已

**回答：**同学，你仔细看下文章，文章后面对代码做了改动的，他是两次触发了gc

**问题：**

G1的垃圾回收器，存不存在类似 P+CMS频繁的回收进而导致系统变慢？

看文章资料，G1变为单线程是因为region内存不够导致的。如果频繁的回收，那么内存应该是足够的，并且标记速度也很快。

回答：

g1可能会频繁回收，但是他每次回收时间可控，所以不会对系统造成太大影响

学员思考题回答：

代码改写如下：

1、不改变年轻代的大小，改变SurvivorRatio=4，这样Eden区6912K，两个Survivor有1664K，避免一开始未知对象+128K超过S区的50%，触发动态年龄计算。

2、代码如下：

```
1 public static void main(String[] args) {
2
3     // TODO Auto-generated method stub
4     int age = 16;
5     byte[] array2 = new byte[128*1024];
6
7     while (age >= 0) {
8
9         byte[] array1 = new byte[2*1024*1024];
10        array1 = new byte[1024*1024];
11        array1 = new byte[1024*1024];
12        array1 = null;
13        byte[] array3 = new byte[2*1024*1024]; age--;
14
15    }
16 }
```

实际运行中，发现第15次gc后，日志输出：

```
[ParNew: 7223K->0K(8576K), 0.0011635 secs] concurrent mark-sweep generation total 10240K, used 675K
[0x00000000ff600000, 0x0000000100000000, 0x0000000100000000)
```

**证明15岁之后，array2进入老年代。**

回答：很棒，优秀回答

**问题：**

下面是我的jdk版本：

Java HotSpot(TM) Client VM (25.121-b13) for windows-x86 JRE (1.8.0\_121-b13), built on Dec 12 2016 18:17:00 by "java\_re"  
with MS VC++ 10.0 (VS2010)

为什么我两次例子的GC日志都只有Metaspace，没有class space的消息？

Metaspace used 1709K, capacity 2242K, committed 2368K, reserved 4480K

**回答：**不同的jdk版本，细节略有不同，核心原理一致就行

**学员做思考题之后的总结：**

昨天实验时就发现，虽然设置了对大对象的标准10M，此时分配了8.2M的对象大于8M的Eden区，连YoungGC都不用触发，也不会判断是否符合大对象的标准（8.2M小于10M），直接进入老年代。当然在实际生产中不太可能出现比Eden区还大的对象。

**学员实践总结：**

这次参数和老师的一样了：

```
0.114: [GC (Allocation Failure) 0.114: [ParNew: 7420K->805K(9216K), 0.1442133 secs] 7420K->805K(19456K), 0.1443909
secs] [Times: user=0.00 sys=0.00, real=0.14 secs] 0.260: [GC (Allocation Failure) 0.260: [ParNew: 7109K->0K(9216K),
0.0061104 secs] 7109K->792K(19456K), 0.0061938 secs] [Times: user=0.03 sys=0.02, real=0.01 secs] Heap par new
generation total 9216K, used 2212K [0x00000000fec00000, 0x00000000ff600000, 0x00000000ff600000) eden space 8192K,
27% used [0x00000000fec00000, 0x00000000fee290e0, 0x00000000ff400000) from space 1024K, 0% used
[0x00000000ff400000, 0x00000000ff400000, 0x00000000ff500000) to space 1024K, 0% used [0x00000000ff500000,
0x00000000ff500000, 0x00000000ff600000) concurrent mark-sweep generation total 10240K, used 792K
[0x00000000ff600000, 0x0000000100000000, 0x0000000100000000)
```

因为我没有用eclipse或者idea，而是用命令行直接运行的

**回答：**很棒，就是要多去动手实践

**问题：**

老师，有个疑问。为什么G1垃圾收集器需要两个S区呢？而不是把两个S区合并为一个S区，那S区岂不是会更大（这样不就可以更大程度上避免对象进入老年代）？

既然采用的是复制算法，那么只用一个S区应该也是可以的吧，在每次新生代gc之后，把回收的 Survivor Region 复制到另一个空闲的，没人用的Region不就好了吗？

**回答：**

如果只有一个S区的话，假设Eden和S现在对象都放满了，需要垃圾回收，此时复制算法需要先把存活对象标记出来，放入一个地方，那此时没有一个空白区域给他放，怎么办呢？所以必须要两个S区来使用